模型评价与模型改善

CS245　数据科学基础　陆朝俊
叶泽林 515030910468

1　问题描述

给定一个机器学习问题（如分类）以及相应的数据集，可以训练出一个初步的模型，但在投入实际应用前，需要对该模型的性能进行评价甚至改善。一般地，通过组合模型的方式，可以提高模型的效果。这次作业中，我将在自己建立的分类数据集上探索决策树模型单独工作以及组合工作时的性能。

2　解决方案[1]

2.1　数据集的建立及模型的选择

由于现有的中小型数据集存在一些难以满足实验要求的属性，我选择自己建立一个分类数据集 D（后文均以 D 表示此数据集），之后的所有实验均基于此数据集上进行。数据集 D 包含 500 个样本，30 个特征（存在冗余和重复的特征），为更好地看出模型性能的区别，我在 D 中添加了部分噪声。关于数据集 D 详细的设定参数可参考附录 A.1。

解决分类问题的机器学习模型主要有逻辑回归、决策树以及 KNN，在这次作业中，我选择决策树作为探索的模型。

2.2　探索模型单独工作的性能

选定数据集和模型后，我首先探索单独的决策树模型在数据集 D 上的性能，探索方法如下：

1. **直接划分训练集和测试集**

   在一般应用中，对模型性能最朴素的评价方法即为：将数据集划分为训练集和测试集，在训练集上训练模型后，根据模型在测试集上的表现评价模型性能。因此，我首先采用这个方法对模型的性能进行一个粗略的估计。

2. **分层交叉验证**

   从理论上说，仅凭在一个训练集 D_1 上训练出的模型 M_1 无法代表模型 M（在我的实验中为决策树模型）解决这个问题的效果。分层交叉验证正好解决了这一问题，其步骤大致如下：

   (a) 将数据随机等分为 k 个不相交子集 $D_1, D_2, ..., D_k$（每个子集的类分布与初始数据近似相同）；

   (b) 总共执行 k 次训练与测试，在第 i 折时，使用 D_i 作为测试集, 其他子集作为训练集；

   (c) 模型的性能由 k 次迭代的平均效果决定。

   分层交叉验证相当于在同一个数据集上进行了多次训练集-测试集划分。和简单的训练集-测试集划分相比，能更准确地评估模型来自样本的误差。

[1]本次作业的所有代码实现可参见附录 A.3

3. **网格搜索**

网格搜索是通过蛮力试验不同参数组合，从而选择最佳模型的方法。理论上来说，网格搜索与分层交叉验证相结合，可以找出模型解决给定问题最合适的参数，但这一过程需要花费大量的时间和计算资源，因此这次作业中我仅实现了一小部分参数空间上的网格搜索。

2.3 探索模型组合工作的性能

一般地，从一系列模型 $M_1, M_2, ..., M_k$ 创建组合模型 M^*，可以有效提高原模型的效果。组合方法有 bagging、boosting、voting 和 stacking 等，本次作业我探索了 bagging 和 boosting 两种组合方法的效果。Bagging 方法使用并行思想，M^* 通过整合各原模型的结果得到输出；Boosting 方法则采用串行思想，不断利用模型 M_{i-1} 的训练结果训练模型 M_i，以最小化目标函数（损失函数），直到 M_k，属于一种迭代搜索算法。

3 结果展示

3.1 模型单独工作的性能

3.1.1 直接划分训练集和测试集

首先，我将原数据按照一定比例划分为训练集和测试集，划分时保证每一类型的数据在训练集和测试集的占比大致服从原分布，再执行一般的“训练集上训练-测试集上测试”流程。测试结果如表 1 所示。

表 1: 不同比例的训练集-测试集划分下，决策树模型在 D 上的性能

测试集占比（其中 0 表示随机分类）	测试集上平均精确度	测试集上平均召回率	测试集上平均 f1-score	训练集上平均 f1-score
0	0.38	0.36	0.36	0.33
0.05	0.78	0.72	0.73	1.00
0.1	0.59	0.56	0.56	1.00
0.2	0.57	0.56	0.56	1.00
0.3	0.57	0.56	0.56	1.00
0.4	0.56	0.56	0.55	1.00
0.5	0.50	0.48	0.48	1.00
0.8	0.49	0.48	0.48	1.00

粗略测试的结果表明：不论训练集-测试集的划分比例如何，该模型总是趋于过拟合，对新数据的分类效果较差。

3.1.2 分层交叉验证

上小节仅对决策树模型在 D 上单独工作的性能进行了粗略的估计，下面使用分层交叉验证对模型单独工作的性能做更完整的探索，结果参见图 1。

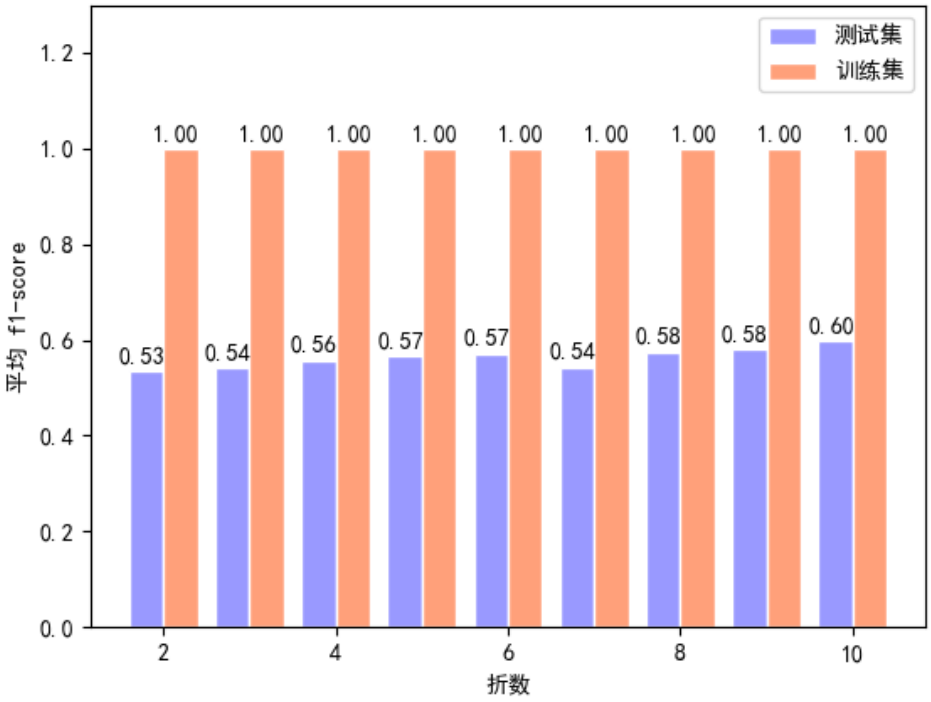


图 1: 分层交叉验证探索模型在 D 上的性能

从图中可以看出，折数和在测试集上的 f1-score 呈正相关（折数越多，训练集则越大，模型对 D 学习得更充分），但均在 0.5 与 0.6 之间，对于一般应用来说也属于较低水平。除此之外，在训练集上的高 f1-score 也说明模型的过拟合问题尤为严重。

3.1.3 网格搜索

这部分实验中，我结合网格搜索和分层交叉验证（10 折），得到决策树模型在 D 上的最佳效果。由于遍历所有参数需要消耗大量的时间和计算资源，我在实验中仅选择了模型的部分重要参数进行网格搜索。具体结果可参见表 2。

表 2: 网格搜索结果

参数名	参数含义	参数范围	模型效果最佳时的值
criterion	特征选择的度量标准	gini, entropy	gini
max_depth	树的最大深度	正整数	9
max_features	寻求最佳划分时 要考虑的特征数目	总特征数或其平方根 或其以 2 为底的对数值	总特征数
presort	是否对数据预先排序	True, False	True
splitter	结点划分策略	best, random	best

经过对上表参数值的网格搜索，得到的模型在分层交叉验证中各测试集的平均 f1-score 为 0.61，虽然只遍历了一部分参数空间，但这个结果在一定程度上也能够代表决策树模型在 D 上分类的最佳性能。最佳模型的可视化结果可参见附录 A.2。

3.1.4 结果分析

通过上面三个小节的探索，可以得出结论：决策树模型单独工作时，在数据集 D 上分类的 f1-score 基本在 0.5-0.6 之间，最高仅达 0.61，而在训练集上的分类效果极佳，表明该模型在 D 上单独工作时容易趋于过拟合，对新数据的分类效果较差，即低偏差、高方差。

3.2 模型组合工作的性能

这一节实验主要探索决策树模型在 bagging 和 boosting 两种组合方式下在 D 上的分类性能。

3.2.1 Bagging

Bagging 方法整合各子模型的输出从而得到结果，因此每个子模型的训练方式对最终结果也有较大的影响。一般地，在训练子模型时选择合适的特征比例（即舍去训练数据的部分特征）可以降低过拟合程度，得到良好的模型。这部分实验中，我选择了**不同子模型个数**以及**不同比例的特征**进行训练，之后采用 10 折分层交叉验证对 bagging 整合后的模型进行性能评价，结果可参见图 2。

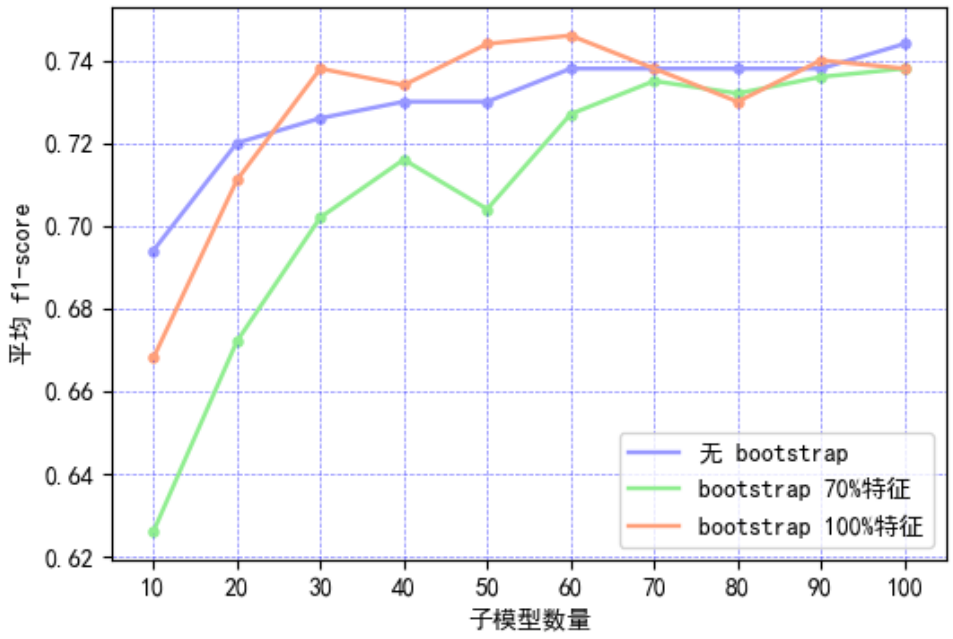
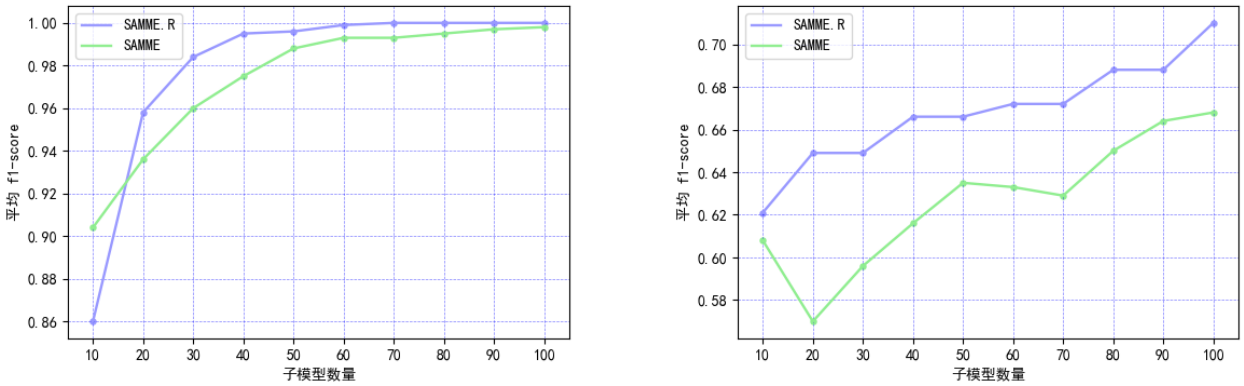


图 2: Bagging 集成模型在测试集上的性能（在训练集上的 f1-score 均为 1，因此不再画出）

对比图 1 和图 2 的结果，可以发现经过 bagging 集成的模型在测试集上能够达到更高的 f1-score（0.74 左右），减小了模型的方差。但 bagging 集成后的模型仍在一定程度上存在过拟合的问题，相关讨论可见 3.2.3 小节。

3.2.2 Boosting

Boosting 不断地使用子模型弥补前一个子模型的“不足”，串行地构造一个强模型。具体实现主要有 Adaboost、SAMME 和 SAMME.R 等。这部分实验中，我使用 SAMME 和 SAMME.R 算法探索了在不同子模型个数下，boosting 集成模型在 10 折分层交叉验证下的性能，结果参见图 3。



(a) 训练集　　(b) 测试集

图 3: Boosting 集成模型性能

从上图可以看出，和模型单独工作相比，boosting 方法同样提升了模型的性能，尤其是降低了模型的方差。对比图 2 及图 3(b) 的结果可知，bagging 和 boosting 在测试集上平均 f1-score 的变化趋势和最终数值基本一致。但结合 boosting 在训练集上的效果（图 3(a) ）分析，boosting 相对于单独模型和 bagging 方法，过拟合程度更低，保证模型低偏差的同时减小了模型的方差。

3.2.3 结果分析

Bagging 和 boosting 虽然都属于组合模型的方法，都能够改进模型的性能，但其背后不同的思想决定了它们对模型的改进有着不同的侧重点。

Bagging 充分利用了不同子模型的预测结果，有利于模型的互补，通过**降低模型方差**改善性能；Boosting 则通过多步迭代，不断优化目标函数，将弱模型逐步转化成强模型，因此和 bagging 相比，boosting 一般能得到**更高的准确度**（即更低的偏差），但也存在模型过拟合的风险（即更高的方差）。

然而，3.2 和 3.2.2 小节的结果与上面的理论分析似乎存在矛盾：Bagging 方法在降低方差上并没有显著的表现，而 boosting 方法的偏差和 bagging 相比，也不存在明显的优势，反而能够在一定程度上降低过拟合程度。

经过分析，我认为这一结果的背后存在多个原因：

1. **参数设置** Bagging 和 boosting 属于不同思想的算法，参数的种类也不同，难以在相同条件下对二者进行比较。二者参数中包含大量连续型参数（如 bagging 中的特征取样比例、boosting 中的学习率），这也使得网格搜索无法实现。因此，对于两种组合算法性能的比较，无法统一在相同的条件下，只能从理论上分析出大致的结果，对具体的问题而言可能各有不同。

2. **子模型的类型** 实验中选择的分类模型是决策树模型，bagging 方法虽然利用不同的训练样本引入随机成分，训练了多个决策树模型，但这些树之间仍存在相关性，尤其是树的上层，几乎具有相同的结构，将这些结构相似的模型进行组合，对方差的降低自然无法体现。而低偏差、高方差，易过拟合也是决策树本身所具有的特点。采用随机森林（分割结点时将特征也做随机选取）算法，可以有效解决这一问题。

3. **数据的影响** 在这次实验中，为了使模型性能的提升更加明显，我在数据集中添加了冗余特征、重复特征甚至噪音（见附录 A.1），加大了分类难度。这些数据上的干扰也在一定程度限制了各种方法的性能，导致实验结果与理论有些许偏差。

4 结论

综上，决策树模型在数据集 D 上单独工作时，偏差较低而方差较大，即在训练集上表现极佳，但在新数据上效果较差。和单独工作相比，模型组合工作时，无论是 bagging 还是 boosting 方法都能够在保证低偏差的情况下有效降低模型的方差，其中 boosting 组合模型的偏差比 bagging 稍高，而方差稍低。但由于决策树模型本身的特点影响，加上数据集存在噪音以及参数未经过大量调整，即使是组合模型也难以满足实际应用的性能需求。

A 附录

A.1 分类数据集 D 的详细信息

表 3: 分类数据集 D 的详细信息

属性	值
样本个数	50
特征个数	30
类个数	3
类分布	每个类均匀分布
包含信息的特征占比	0.6
冗余特征的占比（含信息特征的线性组合）	0.1
重复特征的占比（随机取自含信息特征和冗余特征）	0.1
噪声占比	0.03

A.2 决策树单独工作的最佳模型可视化

由于图片较大，若有需要可放大后详细观看，不影响图片清晰度。

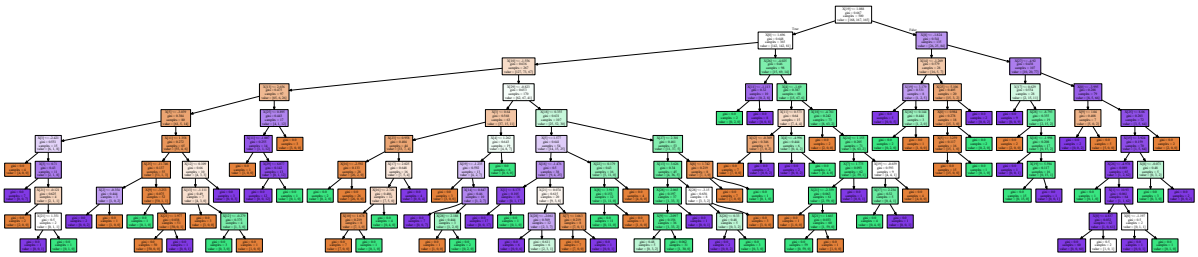


图 4: 决策树单独工作的最佳模型

A.3 主要代码

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']      # display Chinese
plt.rcParams['axes.unicode_minus'] = False        # display minus sign
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA

import os
import time

class bostonAnalyzer(object):
    def __init__(self):
        # load dataset
        self.dataset = datasets.load_boston()
```

```
15          self.data = self.dataset.data
16          self.target = self.dataset.target
17
18          # for plot
19          self.var = []
20          self.t = []
21
22          if not os.path.exists('report/img'):
23              os.makedirs('report/img')
24
25      def run(self, vis=False):
26          '''
27          Run PCA for 1-13 dimensions
28          :param vis: True: plot dim 1-3; False: not plot
29          '''
30
31          self.var.clear()
32          self.t.clear()
33
34          with open('report/result.txt', 'w') as f:
35              for i in range(13):
36                  t = time.time()
37                  pca_op = PCA(n_components=i)
38                  pca_res = pca_op.fit_transform(self.data)
39                  t = time.time() - t
40
41                  self.var.append(pca_op.explained_variance_ratio_.sum()
                        )
42                  self.t.append(t)
43
44                  # write log
45                  f.write('###### Dimension %d ######\n' % i)
46                  f.write(str(pca_res.shape) + '\n')
47                  f.write(str(pca_op.explained_variance_ratio_) + '\n')
48                  f.write(str(pca_op.explained_variance_ratio_.sum()) +
                        '\n')
49                  f.write(str(t) + '\n\n')
50
51                  # visualize for debug & plot
52                  if vis:
53                      if i == 1:
54                          # plot 1 dimension
55                          plt.scatter(pca_res[:, 0], pca_res[:, 0], s
                                =14, c=self.target)
56                          plt.savefig('report/img/pca-%d' % i)
57                          plt.show()
58                      elif i == 2:
59                          # plot 2 dimensions
```

```
60                          plt.scatter(pca_res[:,0], pca_res[:,1], s=8, c
                                =self.target)
61                          plt.savefig('report/img/pca-%d' % i)
62                          plt.show()
63                      elif i == 3:
64                          # plot 3 dimensions
65                          ax = plt.subplot(projection='3d')
66                          ax.scatter(pca_res[:, 0], pca_res[:, 1],
                                pca_res[:, 2], s=8, c=self.target)
67                          plt.savefig('report/img/pca-%d' % i)
68                          plt.show()
69
70      def show(self):
71          '''
72          Show the basic info of Boston dataset & plot k-lines
73          '''
74          print(self.data.shape)
75          print(self.target.shape)
76          self._k_line_radio()
77          self._k_line_time()
78
79      def _k_line_radio(self):
80          '''
81          Plot k-line of variant radio
82          '''
83          x = list(range(len(self.var)))
84          plt.scatter(x, self.var, s=14, c='r')
85          plt.plot(x, self.var)
86          plt.xlabel('主成分个数')
87          plt.ylabel('降维后各特征方差比例之和')
88          plt.savefig('report/img/kline-radio')
89          plt.show()
90
91      def _k_line_time(self):
92          '''
93          Plot k-line of time
94          '''
95          x = list(range(len(self.t)))
96          plt.scatter(x, self.t, s=14, c='r')
97          plt.plot(x, self.t)
98          plt.xlabel('主成分个数')
99          plt.ylabel('PCA算法消耗时间(s)')
100         plt.savefig('report/img/kline-time')
101         plt.show()
102
103 if __name__ == '__main__':
104     bA = bostonAnalyzer()
105     bA.run(True)
```

```
106         bA . show ( )
```